Data Processing with Python /Celery and RabbitMQ in

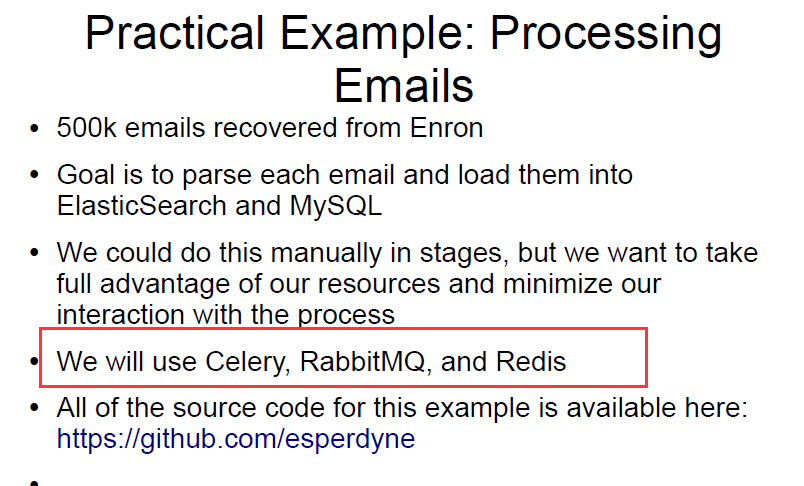
**the Docker environment**

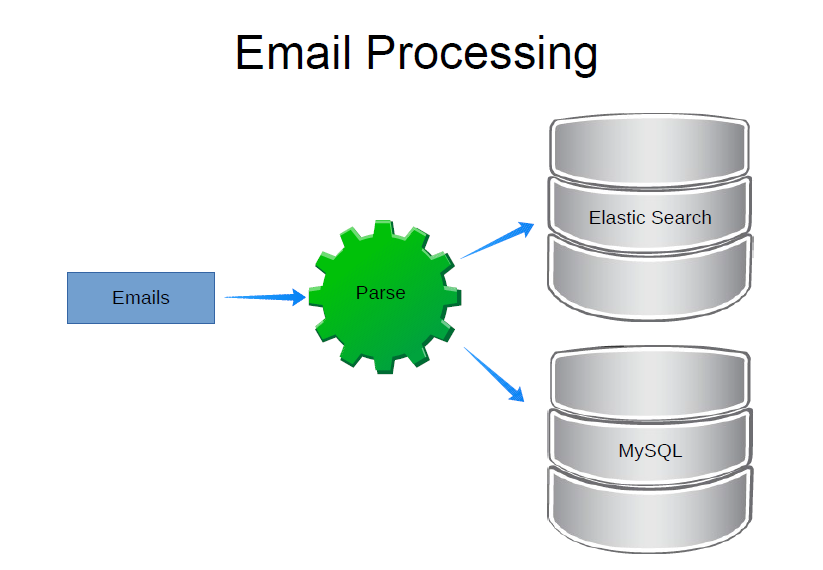
# 1 back ground

Large scale of data processing is a very common task in the modern AI application. And we need a reliable component structure to support such ETL tasks.

## 1.1 use case story

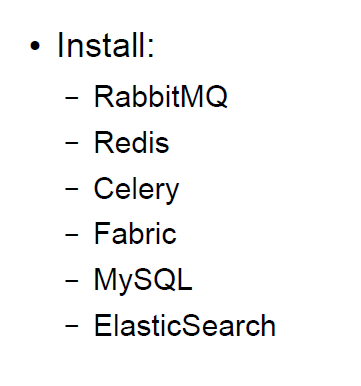
Here is the back ground of the application.





Suppose we have lot’s of email, and we wish to copy the email to the document database Elastic Search and we also wish to copy the email to the rational database MySQL. That is to say we will distribute all the emails to 2 destinations simultaneously.

In order to support such use-cases, we use the following components to support such application.



## 1.2 pain points

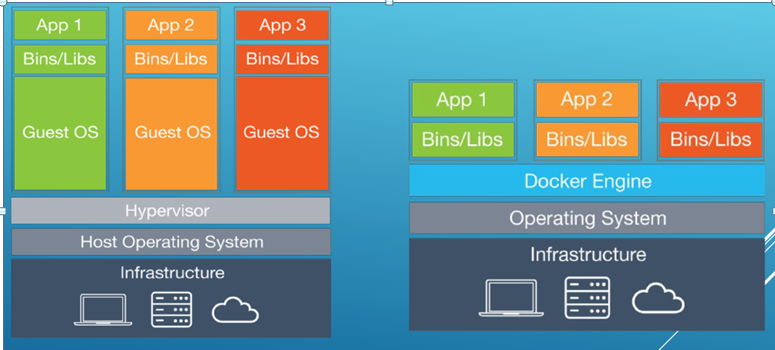
With the help of the github document and relative source code, we have finally done the experiments via a MAC computer first. And then we transfer the application to our formal environment , a computer of 24 CPUs and 96G memories . Yet , the computer locates in a protected area, which could not access to the internet. With so many components to install , that is really a very difficult job.

That is to say, it takes you more that 1 week to install all the required components, and you only spent 1 day to run the experiments. So to prepare for the environment is a hard labour job.

What is more, to copy the information structure of the email to relation database is not so quickly. So maybe we need a faster database.

## 1.3 the docker container

In order to simplify the component preparation, we hope to use the docker as the basic platform.

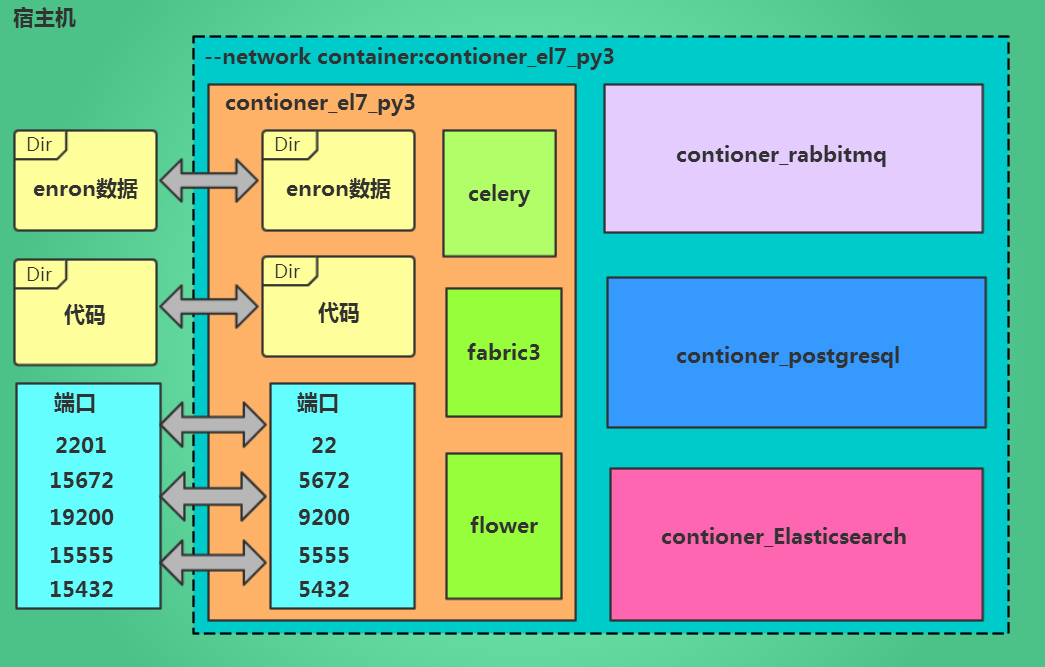


In the docker environment, we will can encapsulate various component in different docker container. In this way, the docker container in your personal computer is exactly the same as the docker container in the formal computer.

Therefore, the docker container is the key element of easy preparing the environment.

# 2、 the Deployment view

## 2.1 Global Deployment view



From the diagram, we have 4 docker containers in application; Which can be described below.

1, **celery container**: it is used to handle the distributed task queue and the main processing component.

2**, rabbitmq container**: it is used to handle the rabbitMQ message service.

3,**postgressql container**: it is used to handle the database persistence service.

4,**ElasticSearch container**: it is used to handle the document database persistence service.

And we have the following interworking port:

|  |  |  |
| --- | --- | --- |
| **id** | **port** | **service** |
| **1** | **5432** | **postgressql** |
| **2** | **9200** | **Elasticsearch** |
| **3** | **5672** | **rabbitmq** |
| **4** | **22** |  |
| **5** | **5555** |  |
| **6** | **3306** |  |

## 2.2 Docker environment

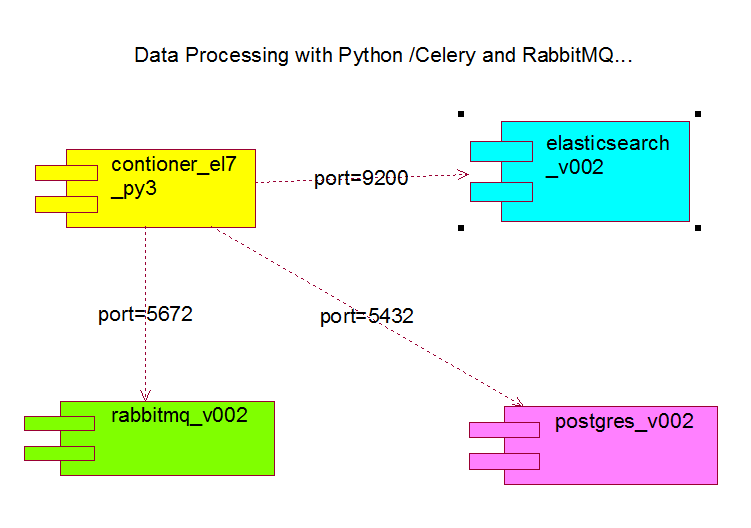
So we must first install the Docker environment to support the overlaying component!

1. /\*uninstall the old version\*/
2. $ sudo yum remove docker \
3. docker-client \
4. docker-client-latest \
5. docker-common \
6. docker-latest \
7. docker-latest-logrotate \
8. docker-logrotate \
9. docker-selinux \
10. docker-engine-selinux \
11. docker-engine
13. /\*install the dependency package\*/
14. $ sudo yum install -y yum-utils \
15. device-mapper-persistent-data \
16. lvm2
18. /\*install the source\*/
19. $ sudo yum-config-manager \
20. --add-repo \
21. https://download.docker.com/linux/centos/docker-ce.repo
23. /\*install the latest CE version\*/
24. $ sudo yum install docker-ce

## 2.3 Docker running

Then we will call the Docker installation application.

1. Docker image pull：
2. docker pull docker.io/sky46821/centos7-python3.6
3. docker pull docker.io/postgres
4. docker pull docker.io/elasticsearch
5. docker pull docker.io/rabbitm
6. Create each component container：



1. /\*to create the master container ,to run celery、fabric、flower\*/
2. docker run -it \
3. --name "contioner\_el7\_py3 " \
4. --publish 2201:22 \
5. --publish 15672:5672 \
6. --publish 19200:9200 \
7. --publish 15555:5555 \
8. --publish 15432:5432 \
9. --publish 13306:3306 \
10. -v /data:/data \
11. -v /app:/app \
12. --user root \
13. sky46821/centos7-python3.6 \
14. /bin/bash
16. /\*to create the rabbitmq container, default port 5672\*/
17. docker run -it \
18. --name "rabbitmq\_v002 " \
19. --network container:contioner\_el7\_py3 \
20. -d rabbitmq
22. /\*to create the elasticsearch container, default port 9200\*/
23. docker run -it \
24. --name "elasticsearch\_v002 " \
25. --network container:contioner\_el7\_py3 \
26. -e "http.host=0.0.0.0" \
27. -e "transport.host=127.0.0.1" \
28. -d elasticsearch \
29. elasticsearch
31. /\*to create postgres, default port 5432\*/
32. docker run \
33. --name "postgres\_v002 " \
34. -e POSTGRES\_PASSWORD="test" \
35. --network container:contioner\_el7\_py3 \
36. -d postgres
37. To install the required component in the **master container**：
38. docker exec -it contioner\_el7\_py3 /bin/bash
39. yum -y install git
40. pip3 install celery
41. pip3 install fabric3
42. pip3 install flower
43. pip3 install sqlalchemy
44. pip3 install psycopg2
45. pip3 install elasticsearch
46. pull skeleton code：
47. git clone https://github.com/esperdyne/celery-message-processing.git

Directory structure：

1. celery-message-processing(root directory)
2. ├── celery-logs（log direcory）
3. │   ├── celery.log（celery running log）
4. │   ├── db\_deploy.log（task-db\_deploy running log）
5. │   ├── es\_deploy.log（task-es\_deploy running log）
6. │   └── parse.log（task-parse running log）
7. ├── celery-pids（pid directory）
8. │   ├── celery.pid(celery-pid file)
9. │   ├── db\_deploy.pid(task-db\_deploy-pid file )
10. │   ├── es\_deploy.pid(task-es\_deploy-pid file)
11. │   └── parse.pid(task-parse-pid file)
12. ├── fabfile.py（fabric-python program file）
13. ├── maildir -> /data/enron/maildir/（enron data directory）
14. ├── proj
15. │   ├── celery.py（celery-python program file）
16. │   ├── \_\_init\_\_.py
17. │   └── tasks.py（define task-python program file）
18. └── README.md

## 2.4 program adaption

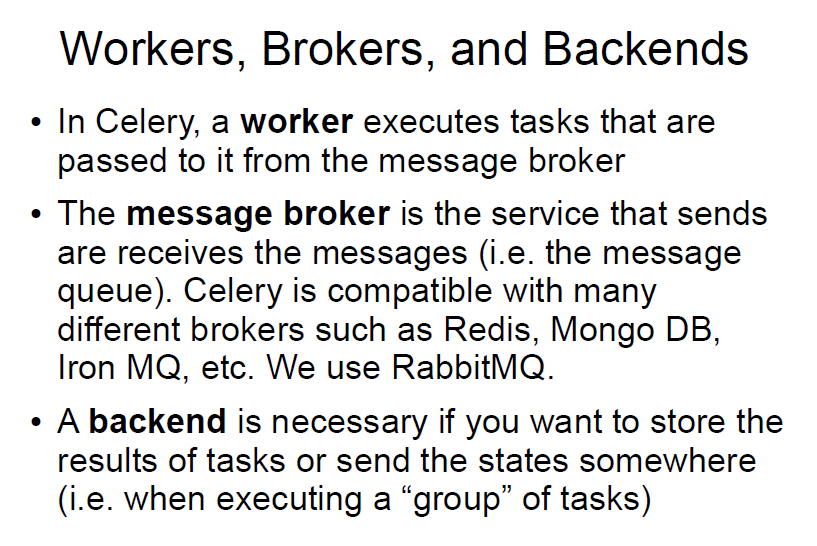
### 2.4.1 celery.py

To configure the celery.py arguments：

1. app = Celery('proj',
2. broker='**amqp://localhost**',
3. backend='**amqp://localhost**',
4. include=['proj.tasks'])

here the broker='**amqp://localhost**' represent that the broker service choose the rabbit mq.

And the backend='**amqp://localhost**' represent that the backup service choose the rabbit mq also



### 2.4.2 tasks.py

To configure tasks.py arguments：

1. **def** \_init\_database(self):
2. """Set up the PG database"""
3. db = create\_engine('**postgresql://messages:test@localhost:5432/messages**')
4. metadata = MetaData(db)
5. messages\_table = Table('messages', metadata,
6. Column('message\_id', String(255), primary\_key = True),
7. Column('subject', String(255)),
8. Column('to', String(255)),
9. Column('x\_to', String(255)),
10. Column('from', String(255)),
11. Column('x\_from', String(255)),
12. Column('cc', String(255)),
13. Column('x\_cc', String(255)),
14. Column('bcc', String(255)),
15. Column('x\_bcc', String(255)),
16. Column('payload', Text()))
17. messages\_table.create(checkfirst=True)
18. self.\_messages\_table = messages\_table

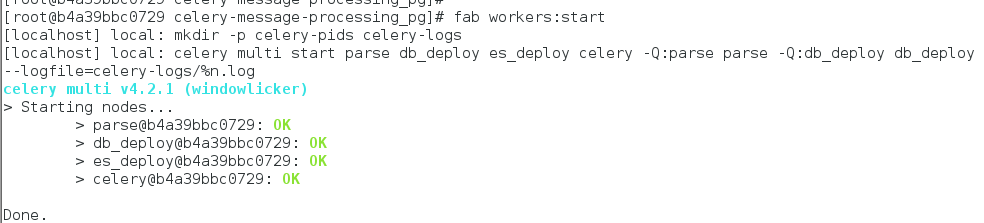
 db = create\_engine('**postgresql://messages:test@localhost:5432/messages**')

here we choose the postgressql as the basic DB persistence service.

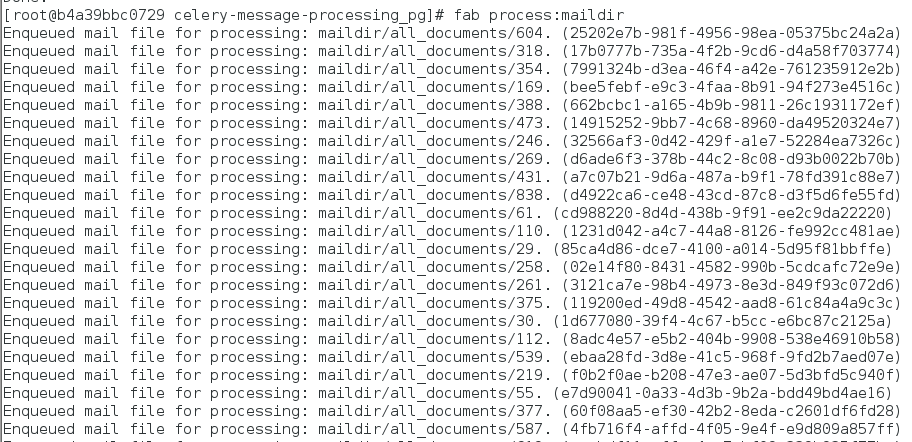
## 2.5 test case

When everything is ready, we’ll test the application.

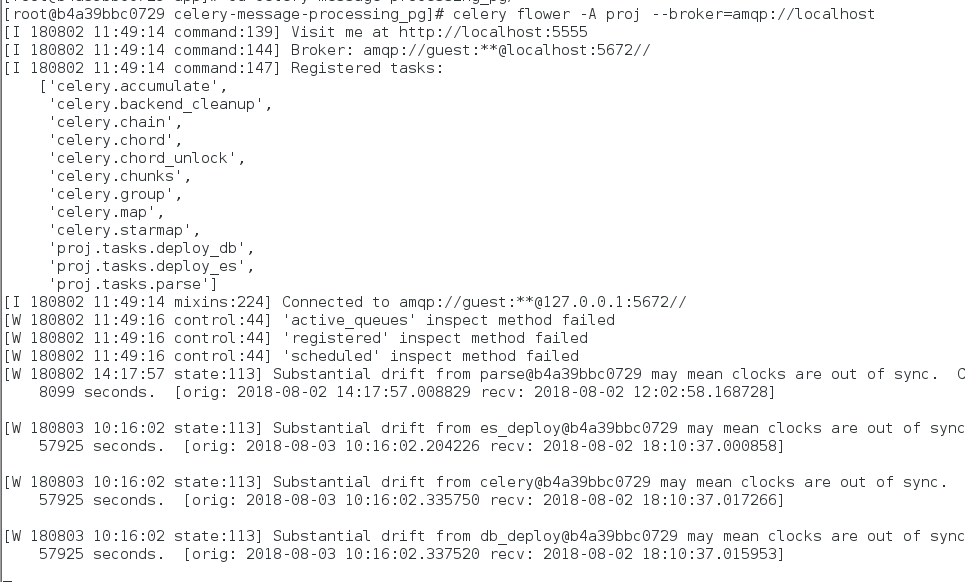
### 2.5.1 start the celery：



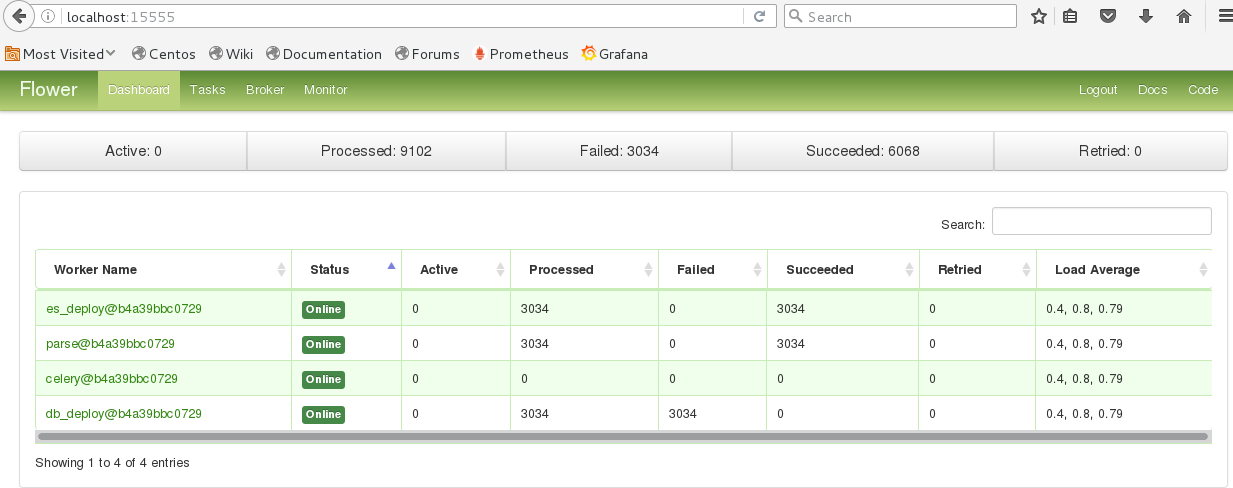
### 2.5.2 start the fabric：



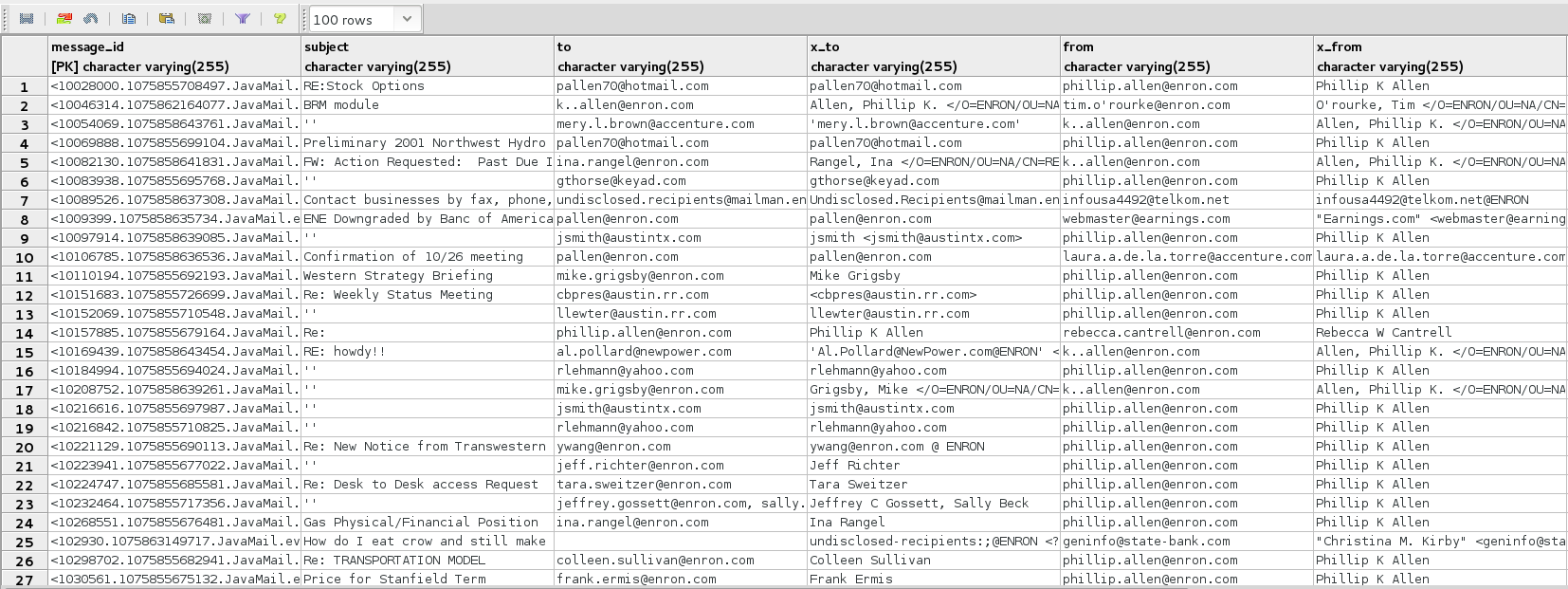
### 2.5.3 start the Flower



**Flower monitor view：**



### 2.5.4 Postgresql table persistence：



### 2.5.5 Elasticsearch table persistence ：



## 2.6 test result

### 2.6.1 executing environment

**1,** enron email list：

**Size of the email**：2.6G

**Number of the file**：517401

2 hardware dimension：

CPU i7-6700HQ 4 cores

memeory 6GB

3**：hardware payload**

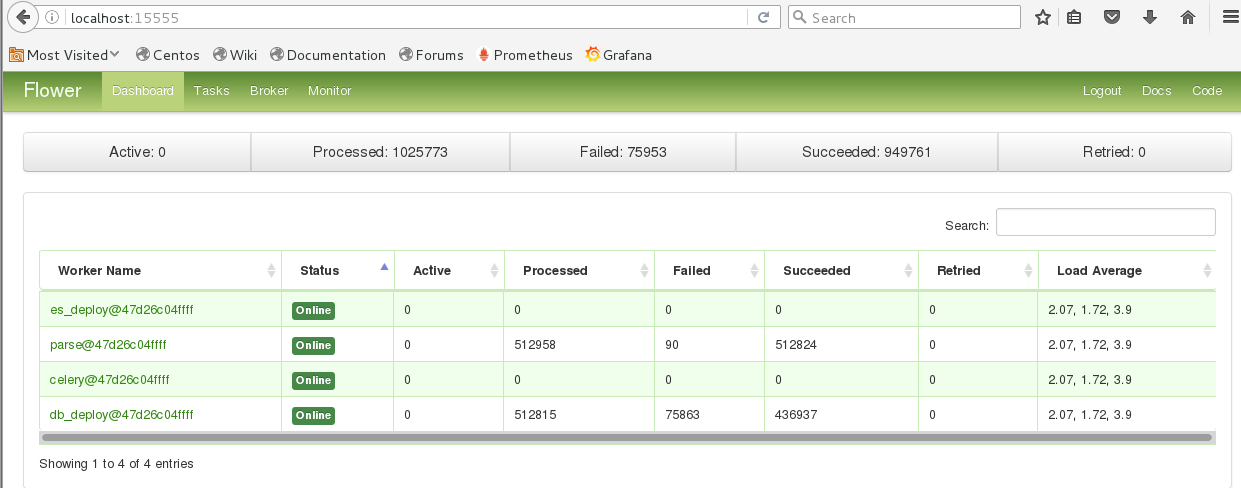
CPU payload ：360% （1 core metric 100%）

Memory payload：4GB

**4:Processing result：**

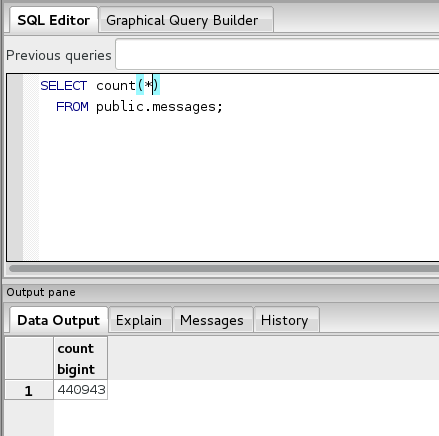
Total processing time：68 minutes15:25-16:33）

Number of file handled 1 minute：7500



5 ,record in the database

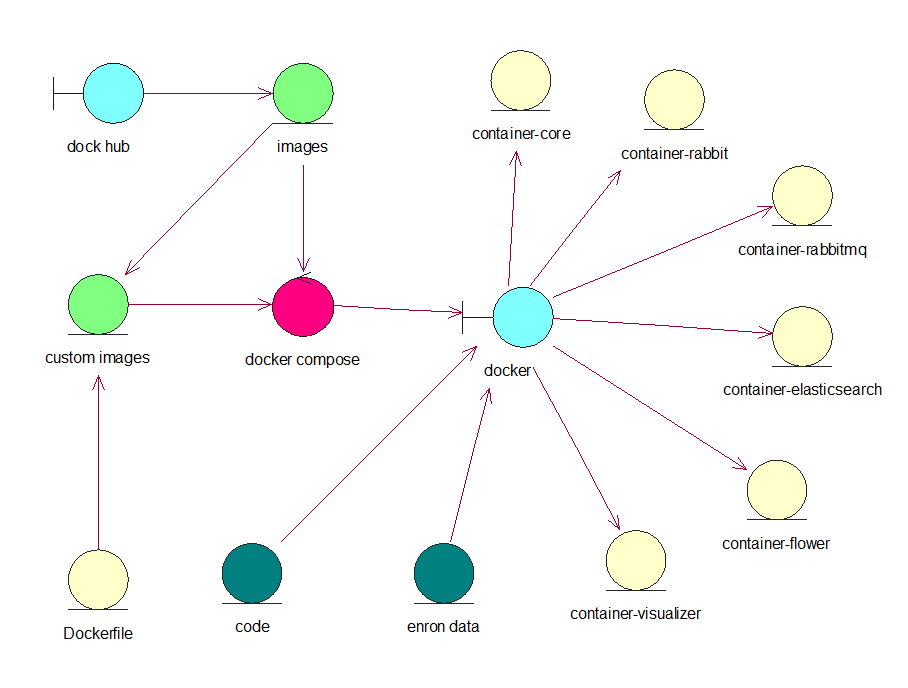
**Number of the file in DB**：440943



## 2.7 docker-swarm based docker

if we wish to simplify the installation procedure of the docker environment, we could future use the YML to configure the whole process.

Deploy procedure：



1. Docker image pull：
2. docker pull docker.io/sky46821/centos7-python3.6
3. docker pull docker.io/postgres
4. docker pull docker.io/elasticsearch
5. docker pull docker.io/rabbitm

The main images can be downloaded from the Docker HUB.

1. Utilize the **docker compose/docker stack** to automatic deploy

Dockerfile:

1. FROM sky46821/centos7-python3.6
3. # Install any needed packages specified in requirements.txt
4. RUN pip3 install -r requirements.txt

Requirement.txt（to define the module which need to install定义需要安装的模块）

1. celery
2. fabric3
3. sqlalchemy
4. psycopg2
5. elasticsearch
6. redis

docker-compos.yaml file：

1. version: "3"
2. services:
3. postgres:
4. image: postgres
5. hostname: postgres
6. restart: always
7. ports:
8. - "15432:5432"
9. volumes:
10. - "/etc/localtime:/etc/localtime:ro"
11. deploy:
12. replicas: 1
13. placement:
14. constraints: [node.role == manager]
15. restart\_policy:
16. condition: on-failure
17. environment:
18. POSTGRES\_USER: "messages"
19. POSTGRES\_PASSWORD: "test123456"
21. rabbitmq:
22. image: rabbitmq
23. hostname: rabbitmq
24. restart: always
25. ports:
26. - "15672:5672"
27. volumes:
28. - "/etc/localtime:/etc/localtime:ro"
29. deploy:
30. replicas: 1
31. placement:
32. constraints: [node.role == manager]
33. restart\_policy:
34. condition: on-failure
36. redis:
37. image: redis
38. hostname: redis
39. restart: always
40. ports:
41. - "16379:6379"
42. volumes:
43. - "/etc/localtime:/etc/localtime:ro"
44. deploy:
45. replicas: 1
46. placement:
47. constraints: [node.role == manager]
48. restart\_policy:
49. condition: on-failure
51. elasticsearch:
52. image: elasticsearch
53. hostname: elasticsearch
54. restart: always
55. ports:
56. - "19200:9200"
57. volumes:
58. - "/etc/localtime:/etc/localtime:ro"
59. deploy:
60. replicas: 1
61. placement:
62. constraints: [node.role == manager]
63. restart\_policy:
64. condition: on-failure
66. flower:
67. image: totem/celery-flower-docker
68. hostname: flower
69. restart: always
70. ports:
71. - "15555:5555"
72. volumes:
73. - "/etc/localtime:/etc/localtime:ro"
74. deploy:
75. replicas: 1
76. placement:
77. constraints: [node.role == manager]
78. restart\_policy:
79. condition: on-failure
80. environment:
81. FLOWER\_BASIC\_AUTH: "root:test123456"
83. visualizer:
84. image: dockersamples/visualizer
85. hostname: visualizer
86. restart: always
87. ports:
88. - "18080:8080"
89. volumes:
90. - "/var/run/docker.sock:/var/run/docker.sock"
91. - "/etc/localtime:/etc/localtime:ro"
92. deploy:
93. replicas: 1
94. placement:
95. constraints: [node.role == manager]
96. restart\_policy:
97. condition: on-failure
99. core-container:
100. image: core-container
101. hostname: core-container
102. volumes:
103. - "/data:/data"
104. - "/app:/app"
105. - "/etc/localtime:/etc/localtime:ro"
106. ports:
107. - '10022:22'
108. deploy:
109. replicas: 1
110. placement:
111. constraints: [node.role == manager]
112. restart\_policy:
113. condition: on-failure
114. networks:
115. **default**:
116. driver: overlay

deploy script：

1. docker stack deploy -c ./docker-compose.yaml STACK